Second Competition on Software Verification* (Summary of SV-COMP 2013)

Dirk Beyer

University of Passau, Germany

Abstract. This report describes the 2nd International Competition on Software Verification (SV-COMP 2013), which is the second edition of this thorough evaluation of fully automatic verifiers for software programs. The reported results represent the 2012 state-of-the-art in automatic software verification, in terms of effectiveness and efficiency, and as available and participated. The benchmark set of verification tasks consists of 2 315 programs, written in C, and exposing features of integers, heap-data structures, bit-vector operations, and concurrency; the properties include reachability and memory safety. The competition is again organized as a satellite event of TACAS.

1 Introduction

Software verification is a major research area within computer science, and modern implementations of verification tools become industrially relevant due to recent advancements in verification technology, e.g., new data structures for abstract domains and efficient solvers for satisfiability modulo theories (SMT). The competition on software verification systematically compares the effectiveness and efficiency of modern software verifiers. The community has gathered a benchmark set of a total of 2 315 verification tasks, which are arranged in eleven categories, according to the characteristics of the programs and the properties to verify. In difference to other competitions [1] [2] [3] [4] [5] [6], SV-COMP [1] focuses on the evaluation of tools for *fully automatic* verification of *source code* programs in a standard programming language. All experiments are performed on dedicated competition machines with a resource specification that is the same for all participants. The goals of the competition on software verification are to:

- present the state-of-the-art in software-verification research,
- establish a widely accepted benchmark set of software verification tasks,
- make modern software verifiers visible, together with their strengths, and
- accelerate the transfer of new technologies to verification practice.

* `http://sv-comp.sosy-lab.org`

[1] `http://www.satcompetition.org`

[2] `http://www.smtcomp.org`

[3] `http://ipc.icaps-conference.org`

[4] `http://www.qbflib.org/competition.html`

[5] `http://fmv.jku.at/hwmcc12`

[6] `http://www.cs.miami.edu/~tptp/CASC`

N. Piterman and S. Smolka (Eds.): TACAS 2013, LNCS 7795, pp. 594–609, 2013.

2 Procedure

The process of the competition consists of three phases: (1) *benchmark submission*, in which new verification tasks are collected and classified into competition categories (as in the first edition, all contributed benchmarks were accepted), (2) *training phase*, in which the benchmark set becomes frozen and teams of the competition candidates inspect the verification tasks and train their tools, (3) *evaluation phase*, in which all competition candidates were applied to the sets of verification tasks and the system descriptions were reviewed by the competition jury (all systems and their descriptions were archived and stamped with SHA hash values), and (4) *approval of verification results*, in which the teams received the preliminary results of their competition candidate. For more details on the procedure, we refer to the previous competition report [1].

3 Definitions and Rules

Verification Tasks. A verification task consists of a C program and a property. A verification run is a non-interactive execution of a competition candidate on a single verification task, in order to check if the following statement is correct: "The program satisfies the property." The result of a verification run is a triple (ANSWER, WITNESS, TIME). ANSWER is one of the following outcomes:

TRUE: The property is satisfied (no path that violates the property exists).
FALSE + Path: The property is violated (i.e., there exists a finite path that violates the property) and a counterexample path is produced and reported.
UNKNOWN: The tool cannot decide the problem, or terminates by a tool crash, or exhausts the computing resources time or memory (i.e., the competition candidate does not succeed in computing an answer TRUE or FALSE).

If the answer is FALSE, then a counterexample path must be produced and provided as WITNESS. There was so far no particular fixed format for the error path. The path has to be written to a file or on screen in a reasonable format to make it possible to check validity. TIME is the consumed CPU time until the verifier terminates. It includes the consumed CPU time of all processes that the verifier starts. If TIME is equal to or larger than the time limit, then the verifier is terminated and the ANSWER is set to 'timeout' (and interpreted as UNKNOWN). The C programs are partitioned into categories, which are defined in category-set files. The categories and the contained programs are explained on the benchmark page of the competition.

Property for Label Reachability. The property to be verified for all categories except 'MemorySafety' is the reachability property p_{error}, which is encoded in the program source code (using the error label 'ERROR'):

p_{error} **:** The C label 'ERROR' is not reachable from the entry of function 'main' on any finite execution of the program.

Table 1. Scoring schema for SV-COMP 2013

Reported result	Points	Description
UNKNOWN	0	Failure to compute verification result.
FALSE correct	+1	Violation of property in program was correctly found.
FALSE incorrect	−4	Violation reported for correct program (false alarm).
TRUE correct	+2	Correct program reported to satisfy property.
TRUE incorrect	−8	Incorrect program reported as correct (missed bug).

Property for Memory Safety. The property to be verified for category 'MemorySafety' is the memory-safety property $p_{\text{mem}-\text{safety}}$, which consists of three partial properties:

$p_{\text{mem}-\text{safety}}$ **:** $p_{\text{valid}-\text{free}} \wedge p_{\text{valid}-\text{deref}} \wedge p_{\text{valid}-\text{memtrack}}$

$p_{\text{valid}-\text{free}}$ **:** All memory deallocations are valid (counterexample: invalid free). More precisely: There exists no finite execution of the program from the entry of function 'main' on which an invalid memory deallocation occurs.

$p_{\text{valid}-\text{deref}}$ **:** All pointer dereferences are valid (counterexample: invalid dereference). More precisely: There exists no finite execution of the program from the entry of function 'main' on which an invalid pointer dereference occurs.

$p_{\text{valid}-\text{memtrack}}$ **:** All allocated memory is tracked, i.e., pointed to or deallocated (counterexample: memory leak). More precisely: There exists no finite execution of the program from the entry of function 'main' on which the program lost track of some previously allocated memory.

If a verification run detects that the property $p_{\text{mem}-\text{safety}}$ is violated, the verification result is required to be more specific; the violated partial property has to be given in the result: FALSE(p), with $p \in \{p_{\text{valid}-\text{free}}, p_{\text{valid}-\text{deref}}, p_{\text{valid}-\text{memtrack}}\}$, means that the (partial) property p is violated. The competition rules define that all programs in category 'MemorySafety' violate at most one (partial) property $p \in \{p_{\text{valid}-\text{free}}, p_{\text{valid}-\text{deref}}, p_{\text{valid}-\text{memtrack}}\}$.

Benchmark Verification Tasks. All verification tasks are available for browsing and download via the public SVN repository of the Competition on Software Verification [7]. The programs were assumed to be written in GNU C (many of them adhere to ANSI C). Compared to SV-COMP 2012, it was not a requirement that the programs are provided in CIL (C Intermediate Language).

Evaluation by Scores and Run Time. Table 1 shows the scoring schema for SV-COMP 2013. Compared to the previous SV-COMP, the negative scores are doubled, in order to make it more difficult to compensate incorrect results by some correct results. The participating competition candidates are ranked according to the sum of points. Competition candidates with the same sum of points are further ranked according to success run time. The success run time for a competition candidate is the total CPU time over all verification tasks for which the competition candidate reported a correct verification result.

[7] `https://svn.sosy-lab.org/software/sv-benchmarks/tags/svcomp13`

As in the last edition, all verification runs were also performed on obfuscated versions of all benchmark programs (renaming of variable and function names; renaming of file name). There was no discrepancy between the verification results obtained from the obfuscated programs and the verification results obtained from the corresponding original program.

Opting-Out from Categories. Every team can submit for every category (including meta categories, i.e., categories that consist of verification tasks from other categories) an opt-out statement. In the results table, a dash is entered for that category; no execution results are reported in that category. If a team participates (i.e., does not opt-out), *all* verification tasks that belong to that category are executed with the verifier. The obtained results are reported in the results table; the scores for meta categories are weighted according to the established procedure. (This means, a tool can participate in a meta category and at the same time opt-out from a sub-category, with having the real results of the tool counted towards the meta category, but not reported for the sub-category.)

Computation of Score for Meta Categories. A meta category is a category that consists of several sub-categories. The score for such a meta category is computed from the normalized scores in its sub-categories. In SV-COMP 2013, there are two meta categories: ControlFlowInteger and Overall. ControlFlowInteger consists of the two sub-categories ControlFlowInteger-MemSimple and ControlFlowInteger-MemPrecise. Overall consists of the sub-categories BitVectors, Concurrency, ControlFlowInteger, DeviceDrivers64, FeatureChecks, HeapManipulation, Loops, MemorySafety, ProductLines, and SystemC.

The *score for a meta category* is computed from the scores of all k contained (sub-) categories using a normalization by the number of contained verification tasks: The normalized score sn_i of a verifier in category i is obtained by dividing the score s_i by the number of tasks n_i in category i ($sn_i = s_i/n_i$), then the sum $\Sigma_{i=1}^{k} sn_i$ over the normalized scores of the categories is multiplied by the average number of tasks per category. [8]

The goal is to reduce the influence of a verification task in a large category compared to a verification task in a small category, and thus, balance over the categories. Normalizing by score is not an option because we assigned higher positive scores for expected-true results and higher negative scores for incorrect results. (Normalizing by score would remove those desired differences.)

Competition Jury. The competition jury consists of the chair and one member of each participating team; the team-representing members circulate every year after the candidate-submission deadline. This committee reviews the competition contribution papers and helps the organizer with resolving any disputes that might occur. Table 2 lists the team-representing members of the jury of 2013.

[8] An example calculation can be found on the web page of the competition.

Table 2. Competition candidates with their system-description references and representing jury members

Competition candidate	Ref.	Jury member	Affiliation
Blast 2.7.1	[22]	Vadim Mutilin	Moscow, Russia
CPAchecker 1.1.10-Explicit	[19]	Stefan Löwe	Passau, Germany
CPAchecker 1.1.10-SeqCom	[24]	Philipp Wendler	Passau, Germany
CSeq 2012-10-22	[12]	Bernd Fischer	Southampton, UK
Esbmc 1.20	[20]	Lucas Cordeiro	Manaus, Brazil
Llbmc 2012-10-23	[11]	Carsten Sinz	Karlsruhe, Germany
Predator 2012-10-20	[10]	Tomas Vojnar	Brno, Czech Rep.
Symbiotic 2012-10-21	[23]	Jiri Slaby	Brno, Czech Rep.
Threader 0.92	[21]	Andrey Rybalchenko	Munich, Germany
Ufo 2012-10-22	[14]	Arie Gurfinkel	Pittsburgh, USA
Ultimate Automizer 2012-10-25	[15]	Matthias Heizmann	Freiburg, Germany

4 Participating Teams

This section briefly introduces the competition candidates (alphabetical order). Table 2 provides an overview of the participating candidates. Below, we list for each competition candidate the achieved (top-three) placements in the categories. The detailed summary of the results is presented in Sect. 5.

Table 3 provides an overview of the technologies and concepts that are used by the various competition candidates. The techniques of counterexample-guided abstraction refinement (CEGAR) [9], bounded model checking [8], and interpolation for discovering new facts to refine an abstract model [16] are used by a total of six tools. Other techniques that are offered by the competition candidates are predicate abstraction [13], construction of an abstract reachability graph (ARG) as proof of correctness [2], lazy abstraction [17], and shape analysis [18]. Only three tools support the verification of concurrent programs.

Blast 2.7.1 [2,22], submitted by *Pavel Shved, Mikhail Mandrykin, and Vadim Mutilin* (Russian Academy of Sciences, Russia), has achieved the placement:

- *Bronze* in DeviceDrivers64

Blast 2.7.1[9] is a software model checker that is based on predicate abstraction [13], CEGAR [9], and the interpolation tool CSIsat [7].

CPAchecker 1.1.10-Explicit [19], submitted by *Stefan Löwe* (University of Passau, Germany), has achieved the following placements:

- *Silver* in ControlFlowInteger
- *Silver* in DeviceDrivers64
- *Silver* in SystemC
- *Silver* in Overall

[9] `http://mtc.epfl.ch/software-tools/blast`

Table 3. Technologies and features that the competition candidates offer

Competition candidate	CEGAR	Predicate Abstraction	Bounded Model Checking	Shape Analysis	ARG-based Analysis	Lazy Abstraction	Interpolation	Concurrency Support
Blast	✓	✓			✓	✓	✓	
CPA-Explicit	✓		✓		✓	✓	✓	
CPA-SeqCom	✓	✓	✓		✓	✓	✓	
CSeq			✓					✓
Esbmc			✓					✓
Llbmc			✓					
Predator				✓				
Symbiotic								
Threader	✓	✓			✓		✓	✓
Ufo	✓	✓	✓		✓	✓	✓	
Ultimate	✓	✓				✓	✓	

CPAchecker 1.1.10-Explicit is based on the verification framework CPAchecker [4][10], which implements the formalism of configurable program analysis (CPA) [3]. The competition candidate uses an explicit-state model-checking approach [6] that integrates abstraction, CEGAR, and interpolation.

CPAchecker 1.1.10-SeqCom [24], submitted by *Philipp Wendler* (University of Passau, Germany), has achieved the following placements:

- *Winner* in Overall
- *Bronze* in BitVectors
- *Bronze* in ControlFlowInteger
- *Bronze* in FeatureChecks
- *Bronze* in HeapManipulation
- *Bronze* in ProductLines
- *Bronze* in SystemC

[10] `http://cpachecker.sosy-lab.org`

CPAchecker 1.1.10-SeqCom is also based on the verification framework CPAchecker and uses a sequential combination of an explicit-value analysis and predicate analysis with adjustable-block encoding [5].

CSeq 2012-10-22 [12], submitted by *Bernd Fischer, Omar Inverso, and Gennaro Parlato* (University of Southampton, UK), has achieved the placement:

- *Silver* in Concurrency

CSeq 2012-10-22[11] is an analyzer that transforms concurrent programs into non-deterministic sequential programs and starts a model-checking for sequential programs for the verification of the property on the resulting programs.

Esbmc 1.20 [20], submitted by *Jeremy Morse, Lucas Cordeiro, Denis Nicole, and Bernd Fischer* (University of Southampton, UK / UFAM, Brazil), achieved:

- *Silver* in BitVectors
- *Silver* in Loops
- *Bronze* in Concurrency
- *Bronze* in MemorySafety
- *Bronze* in Overall

Esbmc 1.20[12] is a bounded model checker that uses a combination of context-bounded symbolic model checking and k-induction for the verification of multi-threaded and single-threaded C programs.

Llbmc 2012-10-23 [11], submitted by *Stephan Falke, Florian Merz, and Carsten Sinz* (Karlsruhe Institute of Technology, Germany), has achieved the following placements:

- *Winner* in BitVectors
- *Winner* in Loops
- *Silver* in FeatureChecks
- *Silver* in Heapmanipulation
- *Silver* in MemorySafety
- *Silver* in ProductLines

Llbmc 2012-10-23[13] is a bounded model checker with a focus on a bit-precise analysis for low-level C code. The tool is based on the Llvm compiler infrastructure, and passes the verification conditions to the SMT solver Stp[14], which supports bit-vectors and arrays.

[11] `http://users.ecs.soton.ac.uk/gp4/cseq-0.1a.zip`
[12] `http://esbmc.org`
[13] `http://baldur.iti.uka.de/llbmc`
[14] `http://sites.google.com/site/stpfastprover`

Predator 2012-10-20 [10], submitted by *Kamil Dudka, Petr Muller, Petr Peringer, and Tomas Vojnar* (Brno University of Technology, Czech Republic), has achieved the following placements:

- *Winner* in FeatureChecks
- *Winner* in Heapmanipulation
- *Winner* in MemorySafety

Predator 2012-10-20[15] is a program analyzer focusing on the verification of C programs with dynamically-linked-list data structures. The abstract domain is based on symbolic memory graphs.

Symbiotic 2012-10-21 [23], submitted by *Jiri Slaby, Jan Strejcek, and Marek Trtík* (Masaryk University, Czech Republic), has participated in the categories ControlFlowInteger-MemPrecise, DeviceDrivers64, FeatureChecks, and SystemC. Symbiotic 2012-10-21 [16] is a program analyzer that combines instrumentation, program slicing, and symbolic execution.

Threader 0.92 [21], submitted by *Corneliu Popeea and Andrey Rybalchenko* (TU Munich, Germany), has achieved the following placements:

- *Winner* in Concurrency

Threader 0.92 [17] is a model checker for multi-threaded C programs that is based on compositional reasoning and supports, in addition to safety, also termination properties.

Ufo 2012-10-22 [14], submitted by *Arie Gurfinkel, Aws Albarghouthi, Sagar Chaki, Yi Li, and Marsha Chechik* (SEI, USA and University of Toronto, Canada), has achieved the following placements:

- *Winner* in ControlFlowInteger
- *Winner* in DeviceDrivers64
- *Winner* in ProductLines
- *Winner* in SystemC
- *Bronze* in Loops

Ufo 2012-10-22 [18] is a verifier that combines numerical data-flow domains with CEGAR, interpolation, and feasibility checks based on bounded model checking.

Ultimate Automizer 2012-10-25 [15], submitted by *Matthias Heizmann et al.* (University of Freiburg, Germany), has participated in the categories ControlFlowInteger-MemPrecise and SystemC. Ultimate Automizer [19] is a verifier that is based on trace abstraction, nested interpolants, and interpolation.

[15] `http://www.fit.vutbr.cz/research/groups/verifit/tools/predator`
[16] `https://sf.net/projects/symbiotic`
[17] `http://www.model.in.tum.de/~popeea/research/threader.html`
[18] `https://bitbucket.org/arieg/ufo/wiki/Home`
[19] `http://ultimate.informatik.uni-freiburg.de/automizer`

Table 4. Quantitative overview over all results — Part 1

Competition candidate Representing jury member Affiliation	**BitVectors** 60 points max. 32 verification tasks	**Concurrency** 49 points max. 32 verification tasks	**ControlFlowInteger** 146 points max. 94 verification tasks	**DeviceDrivers64** 2 419 points max. 1 237 verification tasks	**FeatureChecks** 206 points max. 118 verification tasks	**HeapManipulation** 48 points max. 28 verification tasks
Blast 2.7.1 Vadim Mutilin Moscow, Russia	— —	— —	93 7 100 s	**2 338** 2 400 s	130 42 s	— —
CPAchecker-Explicit Stefan Löwe Passau, Germany	16 86 s	0 0 s	**143** 1 200 s	**2 340** 9 700 s	159 180 s	22 30 s
CPAchecker-SeqCom Philipp Wendler Passau, Germany	**17** 190 s	0 0 s	**141** 3 400 s	2 186 30 000 s	**159** 160 s	**22** 29 s
CSeq 2012-10-22 Bernd Fischer Southampton, UK	— 	**17** 270 s	— 	— 	— 	—
Esbmc 1.20 Lucas Cordeiro Manaus, Brazil	**24** 480 s	**15** 1 400 s	90 17 000 s	2 233 46 000 s	132 86 s	—
Llbmc 2012-10-23 Carsten Sinz Karlsruhe, Germany	**60** 36 s	— 	— 	— 	**166** 250 s	**32** 310 s
Predator 2012-10-20 Tomas Vojnar Brno, Czech Republic	-75 95 s	0 0 s	-27 650 s	0 0 s	**166** 6.0 s	**40** 2.3 s
Symbiotic 2012-10-21 Juri Slaby Brno, Czech Republic	— 	— 	— 	870 230 s	23 11 s	—
Threader 0.92 Andrey Rybalchenko Munich, Germany	— 	**43** 570 s	— 	— 	— 	—
Ufo 2012-10-22 Arie Gurfinkel Pittsburgh, USA	— 	— 	**146** 450 s	**2 408** 2 500 s	74 46 s	—
Ultimate 2012-10-25 Matthias Heizmann Freiburg, Germany	— 	— 	— 	— 	— 	—

Table 5. Quantitative overview over all results — Part 2

Competition candidate Representing jury member Affiliation	**Loops** 122 points max. 79 verification tasks	**MemorySafety** 54 points max. 36 verification tasks	**ProductLines** 929 points max. 597 verification tasks	**SystemC** 87 points max. 62 verification tasks	**Overall** 3 791 points max. 2 315 verification tasks
Blast 2.7.1					
Vadim Mutilin	35	—	652	34	80
Moscow, Russia	550 s		16 000 s	2 600 s	30 000 s
CPAchecker-Explicit					
Stefan Löwe	51	0	655	**61**	**2 030**
Passau, Germany	370 s	0 s	7 300 s	3 500 s	22 000 s
CPAchecker-SeqCom					
Philipp Wendler	50	0	**915**	**58**	**2 090**
Passau, Germany	1 400 s	0 s	3 100 s	1 800 s	41 000 s
CSeq 2012-10-22					
Bernd Fischer	—	—	—	—	—
Southampton, UK					
Esbmc 1.20					
Lucas Cordeiro	**94**	**3**	914	57	**1 919**
Manaus, Brazil	5 000 s	1 300 s	1 200 s	8 500 s	81 000 s
Llbmc 2012-10-23					
Carsten Sinz	**112**	**24**	**926**	49	—
Karlsruhe, Germany	540 s	38 s	3 600 s	1 900 s	
Predator 2012-10-20					
Tomas Vojnar	36	**52**	865	-6	799
Brno, Czech Republic	17 s	61 s	7 500 s	1 400 s	9 700 s
Symbiotic 2012-10-21					
Juri Slaby	—	—	—	0	—
Brno, Czech Republic				0 s	
Threader 0.92					
Andrey Rybalchenko	—	—	—	—	—
Munich, Germany					
Ufo 2012-10-22					
Arie Gurfinkel	**54**	—	**929**	**65**	-208
Pittsburgh, USA	750 s		5 000 s	3 000 s	12 000 s
Ultimate 2012-10-25					
Matthias Heizmann	—	—	—	45	—
Freiburg, Germany				4 800 s	

Table 6. Overview of the top-three verifiers for each category

Rank	Candidate	Score	Run Time	Solved Tasks	False Alarms	Missed Bugs
BitVectors						
1	**Llbmc 2012-10-23**	**60**	36	32		
2	Esbmc 1.20	24	480	27	3	2
3	CPAchecker-SeqCom	17	190	10		
Concurrency						
1	**Threader 0.92**	**43**	570	28		
2	CSeq 2012-10-22	17	270	11		
3	Esbmc 1.20	15	1 400	15	2	
ControlFlowInteger						
1	**Ufo 2012-10-22**	**146**	450	94		
2	CPAchecker-Explicit	143	1 200	92		
3	CPAchecker-SeqCom	141	3 400	91		
DeviceDrivers64						
1	**Ufo 2012-10-22**	**2 408**	2 500	1 228		
2	CPAchecker-Explicit	2 340	9 700	1 180		
3	Blast 2.7.1	2 338	2 400	1 188		
FeatureChecks						
1	**Predator 2012-10-20**	**166**	6.0	98		
2	Llbmc 2012-10-23	166	250	98		
3	CPAchecker-SeqCom	159	160	94		
HeapManipulation						
1	**Predator 2012-10-20**	**40**	2.3	24		
2	Llbmc 2012-10-23	32	310	20		
3	CPAchecker-SeqCom	22	29	13		
Loops						
1	**Llbmc 2012-10-23**	**112**	540	74		
2	Esbmc 1.20	94	5 000	74	1	2
3	Ufo 2012-10-22	54	750	64	10	
MemorySafety						
1	**Predator 2012-10-20**	**52**	61	35		
2	Llbmc 2012-10-23	24	38	21		
3	Esbmc 1.20	3	1 300	10	2	
ProductLines						
1	**Ufo 2012-10-22**	**929**	5 000	597		
2	Llbmc 2012-10-23	926	3 600	595		
3	CPAchecker-SeqCom	915	3 100	583		
SystemC						
1	**Ufo 2012-10-22**	**65**	3 000	51		
2	CPAchecker-Explicit	61	3 500	44		
3	CPAchecker-SeqCom	58	1 800	42		
Overall						
1	**CPAchecker-SeqCom**	**2 090**	41 000	1 987		4
2	CPAchecker-Explicit	2 030	22 000	1 872		4
3	Esbmc 1.20	1 919	81 000	2 094	22	16

5 Results and Discussion

The results in this competition report represent the 2012 state-of-the-art in software verification in terms of effectiveness and efficiency, as available and participated. All presented results were approved by the competing teams.

The verification runs of the competition were (natively) executed on a dedicated unloaded compute server with a 3.4 GHz 64-bit Quad Core CPU (Intel i7-2600K) and a GNU/Linux operating system (x86_64-linux). The machine had 16 GB of RAM, of which exactly 15 GB were made available to the competition candidate. Every verification run had a run-time limit of 15 min. The run time in the tables is given in seconds of CPU time and all measurement values are rounded to two significant digits. One complete competition run of all candidates on all verification tasks required a total of 21 days of non-stop machine time; several such competition runs were necessary.

Tables 4 and 5 show the total quantitative overview. The tools are listed in alphabetical order. In every table cell for competition results, we list the score in the first row and the CPU time for successful runs in the second row. The top-three candidates are indicated by having their score formatted in bold face and in larger font size. The entry '—' means that the competition candidate opted-out from the category. For the calculation of the score and for the ranking, the scoring schema in Table 1 was applied.

Table 6 gives an overview of the top-three candidates for each category. The run time is given in seconds of CPU usage for the verification tasks that were successfully solved. The column 'False Alarms' indicates the number of verification tasks for which the tool reported an error but the program was correct (false positive), and column 'Missed Bugs' indicates the number of verification tasks for which the verifier claims that the program fulfills the property although it actually contains a bug (false negative).

Score-Based Quantile Functions for Quality Assessment. A total of six verifiers participated in the category Overall, for which we can discuss the overall performance over all categories together. (Note that the scores are normalized as described in Sect. 3.) Figure 1 illustrates the competition results using the quantile functions over all benchmark verification tasks. The function graph for a competition candidate yields, with each data point (x, y), the maximum run time y for the n fastest correct verification runs with the accumulated score x of all incorrect results and those n correct results.

This new visualization is helpful in analyzing the different aspects of verification quality, as outlined in the following.

Amount of Successful Verification Work. Results for verification tasks have different value, depending on the 'difficulty' of the verification task and on the correctness of the verification answer. This value is modeled by a community-agreed scoring schema (cf. Table 1). The x-width of a graph in Fig. 1 illustrates the value (amount) of successful verification work that the verifier has done. The verifier Ufo 2012-10-22 is the best verification tool in this respect, because its

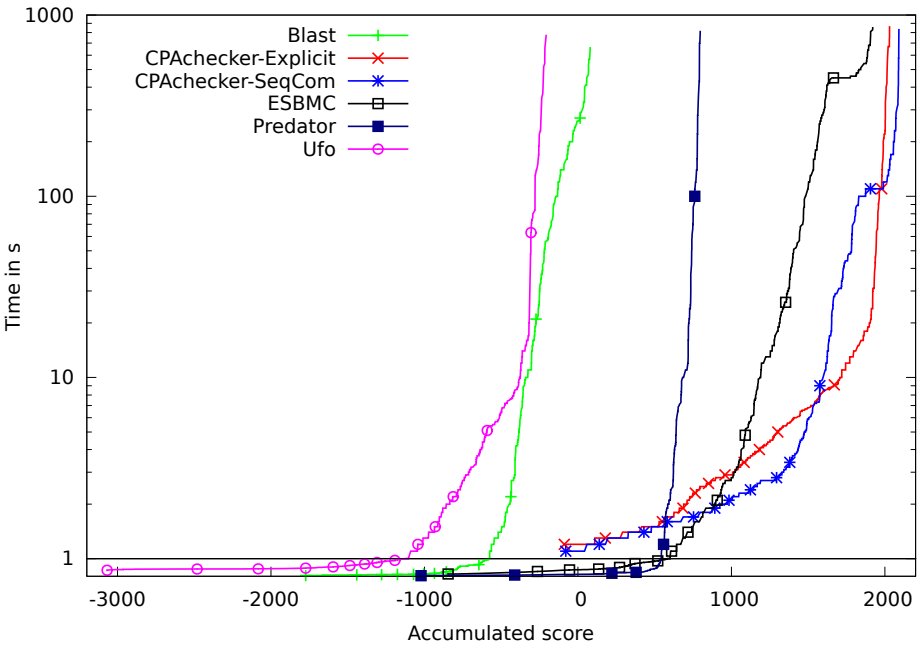

Fig. 1. Quantile functions: For each competition candidate, we plot all data points (x, y) such that the maximum run time of the n fastest correct verification runs is y and x is the accumulated score of all incorrect results and those n correct results. A logarithmic scale is used for the time range from 1 s to 1000 s, and a linear scale is used for the time range between 0 s and 1 s. The graphs are decorated with symbols at every 15-th data point.

quantile function has the largest x-width. This tool solved the most verification tasks, as also witnessed by the large score entries in Tables 4 and 5.

Amount of Incorrect Verification Work. The left-most data point yields the total negative score of a verifier (x-coordinate), i.e., the total score resulting from incorrect verification results. The more right a verifier starts its graph, the less incorrect results it has produced. The two CPAchecker-based candidates start with a very low negative score, and thus, have computed the least value of incorrect results, as also witnessed by the entry for category Overall in Table 6: the verifiers reported wrong results for only 4 out of 2 315 verification tasks.

Overall Quality Measured in Scores. The x-coordinate of the right-most data point of each graph represents the total score of the verification work (and thus, the total value) that was completed by the corresponding competition candidate. This measure identifies the winner of category Overall, as also reported in Table 6 (the x-coordinates match the score values in the table).

Characteristics of the Verification Tools. The y-coordinate of the left-most data point indicates the verification time for the "easiest" verification task for the

verifier, and the y-coordinate of the right-most data point indicates the maximal time that the verifier spend on one single successful task (this is mostly just below the time limit). The area below a graph is proportional to the accumulated run time for all successfully solved verification tasks. Also the shape of the graph can give interesting insights: for example, the graphs for CPAchecker-SeqCom and Esbmc 1.20 show the characteristic bend that occurs if a verifier, after a certain period of time (100 s for CPAchecker-SeqCom and 450 s for Esbmc 1.20), performs a sequential restart with a different strategy.

Robustness, Soundness, and Completeness. The best tools of each category witness that today's verification technology has significantly progressed in terms of overall robustness (avoiding incorrect results), soundness (avoiding false negatives; missed bugs), and completeness (avoiding false positives; false alarms). The last two columns of Table 6 indicate the number of false alarms and missed bugs, respectively, for the top-three verifiers in each category.

6 Conclusion

The second edition of the competition on software verification was again well received in the research community. The participation increased from ten to eleven teams, the benchmark categories increased from seven to eleven, and the total number of benchmarks increased significantly to 2 315 verification tasks, of which 1 805 are expected to be correct, 492 contain a reachable error location, and 18 contain a violation of a memory-safety property. The organizer and the jury were making sure that the competition follows the high quality standards of the TACAS conference, in particular to respect the important principles of fairness, community support, transparency, and technical accuracy.

The results witness a significant progress of the state-of-the-art in developing new concepts for verification of software and in advancing the tool implementations that fully automatically perform the verification. The participating verification tools were able to verify the majority of verification tasks. The top verifiers are quite reliable in the categories that they are focusing on, in terms of robustness, soundness, and completeness. There is no single technique that is superior to all others. The competition candidates —SMT-based model checkers, bounded model checkers, explicit-state model checkers, and program analyzers— showed their different, complementing strength in the various categories.

Acknowledgement. We thank Karlheinz Friedberger for his support during the evaluation phase and for his work on the benchmarking infrastructure.

References

1. Beyer, D.: Competition on Software Verification (SV-COMP). In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 504–524. Springer, Heidelberg (2012)
2. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The Software Model Checker Blast. Int. J. Softw. Tools Technol. Transfer 9(5-6), 505–525 (2007)

3. Beyer, D., Henzinger, T.A., Théoduloz, G.: Program Analysis with Dynamic Precision Adjustment. In: Proc. ASE, pp. 29–38. IEEE (2008)
4. Beyer, D., Keremoglu, M.E.: CPAchecker: A Tool for Configurable Software Verification. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 184–190. Springer, Heidelberg (2011)
5. Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate Abstraction with Adjustable-Block Encoding. In: Proc. FMCAD, pp. 189–197. FMCAD (2010)
6. Beyer, D., Löwe, S.: Explicit-State Software Model Checking Based on CEGAR and Interpolation. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 146–162. Springer, Heidelberg (2013)
7. Beyer, D., Zufferey, D., Majumdar, R.: CSIsat: Interpolation for LA+EUF. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 304–308. Springer, Heidelberg (2008)
8. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
9. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. J. ACM 50(5), 752–794 (2003)
10. Dudka, K., Müller, P., Peringer, P., Vojnar, T.: Predator: A Tool for Verification of Low-level List Manipulation (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 629–631. Springer, Heidelberg (2013)
11. Falke, S., Merz, F., Sinz, C.: LLBMC: Improved Bounded Model Checking of C Programs using LLVM (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 625–628. Springer, Heidelberg (2013)
12. Fischer, B., Inverso, O., Parlato, G.: CSeq: A Sequentialization Tool for C (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 618–620. Springer, Heidelberg (2013)
13. Graf, S., Saidi, H.: Construction of Abstract State Graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
14. Albarghouthi, A., Gurfinkel, A., Li, Y., Chaki, S., Chechik, M.: UFO: Verification with Interpolants and Abstract Interpretation (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 639–642. Springer, Heidelberg (2013)
15. Heizmann, M., Christ, J., Dietsch, D., Ermis, E., Hoenicke, J., Lindenmann, M., Nutz, A., Schilling, C., Podelski, A.: Ultimate Automizer with SMTInterpol (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 643–645. Springer, Heidelberg (2013)
16. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from Proofs. In: Proc. POPL, pp. 232–244. ACM (2004)
17. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy Abstraction. In: Proc. POPL, pp. 58–70. ACM (2002)
18. Jones, N.D., Muchnick, S.S.: A Flexible Approach to Interprocedural Data-Flow Analysis and Programs with Recursive Data Structures. In: POPL, pp. 66–74 (1982)
19. Löwe, S.: CPAchecker with Explicit-Value Analysis Based on CEGAR and Interpolation (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 612–614. Springer, Heidelberg (2013)

20. Morse, J., Cordeiro, L., Nicole, D., Fischer, B.: Handling Unbounded Loops with ESBMC 1.20 (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 621–624. Springer, Heidelberg (2013)
21. Popeea, C., Rybalchenko, A.: Threader: A Verifier for Multi-threaded Programs (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 635–638. Springer, Heidelberg (2013)
22. Shved, P., Mandrykin, M., Mutilin, V.: Predicate Analysis with BLAST 2.7 (Competition Contribution). In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 525–527. Springer, Heidelberg (2012)
23. Slaby, J., Strejček, J., Trtík, M.: Symbiotic: Synergy of Instrumentation, Slicing, and Symbolic Execution (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 632–634. Springer, Heidelberg (2013)
24. Wendler, P.: CPAchecker with Sequential Combination of Explicit-State Analysis and Predicate Analysis (Competition Contribution). In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 615–617. Springer, Heidelberg (2013)